WHAT IS CLAIMED IS:

1. A system for dynamically detecting potential race conditions in a program having a plurality of threads and one or more shared memory locations, the system comprising:

with respect to each shared memory location, (i) a mechanism for maintaining a set of concurrent thread segments that access the location, and (ii) a mechanism for maintaining a first set of locks associated with the location;

with respect to each thread, (i) a mechanism for maintaining a set of thread segments that are ordered before the current thread segment of the thread, and (ii) a mechanism for maintaining a second set of locks that are acquired and released by the thread; and

a mechanism for reporting a warning when a potential race condition is detected.

2. The system of claim 1 wherein the mechanism for maintaining the set of concurrent thread segments comprises a mechanism for maintaining a set of ordered pairs, wherein one member of a pair in the set of ordered pairs is a thread identifier, and the other member of the pair is a virtual clock value associated with the thread identified by the thread identifier.

3. The system of claim 1 wherein the mechanism for maintaining the set of thread segments that are ordered before the current thread segment comprises a mechanism for maintaining a set of ordered pairs, wherein one member of a pair in the set of ordered pairs is a thread identifier, and the other member of the pair is a virtual clock value associated with the thread identified by the thread identifier.

4. A computer-implemented method for dynamically detecting a potential race condition in a program having a plurality of threads and one or more shared memory locations, the method comprising:

with respect to each shared memory location,

maintaining a first set of locks associated with the location, and

maintaining a set of concurrent thread segments that access the location;

with respect to each thread,

maintaining a second set of locks that are acquired and released by the thread, and

maintaining a set of thread segments that are ordered before the current thread segment of the thread.

5. The method of claim 4, further comprising, with respect to each thread, maintaining a virtual clock associated with the thread.

6. The method of claim 5 wherein maintaining the virtual clock comprises initializing the virtual clock to an initial value when the thread is created.

7. The method of claim 6 wherein maintaining the virtual clock comprises initializing the virtual clock to zero when the thread is created.

8. The method of claim 5 wherein maintaining the set of thread segments that are ordered before the current thread segment of the thread comprises maintaining a set of ordered pairs, wherein one member of a pair is a thread identifier, and the other member of the pair is a virtual clock value.

9. The method of claim 8, further comprising, if a first thread forks a second thread:

computing the set of thread segments that are ordered before the current thread segment of the second thread as the union of (a) the set of thread segments that are ordered before the current thread segment of the first thread and (b) a singleton set containing the current thread segment of the first thread;

incrementing the virtual clock associated with the first thread, and

initializing the virtual clock associated with the second thread.

10. The method of claim 9 wherein incrementing the virtual clock associated with the first thread comprises incrementing the virtual clock associated with the first thread by one.

11. The method of claim 9 wherein initializing the virtual clock associated with the second thread comprises initializing the virtual clock associated with the second thread to zero.

12. The method of claim 8, further comprising:

if a first thread joins a forked thread, computing the set of thread segments that are ordered before the current thread segment of the first thread as the union of: (a) the set of thread segments that are ordered before the current thread segment of the first thread, (b) the set containing the thread segments that are ordered before the current thread segment of the forked thread but which do not belong to the first thread, and (c) the singleton set containing the current thread segment of the forked thread.

13. The method of claim 12 wherein the set containing the thread segments that are ordered before the current thread segment of the forked thread but which do not belong to the first thread comprises a set containing the ordered pairs in the set of thread segments that are ordered before the current thread segment of the forked thread, such that the thread identifiers in the ordered pairs do not represent the first thread.

14. The method of claim 8, further comprising, if a thread accesses a shared memory location:

updating the set of concurrent thread segments that access the location by forming a set comprising the union of (a) a set containing the current thread segment of the thread, and (b) a set containing the thread segments in the set of concurrent thread segments that continue to access the location; and

if the updated set of concurrent thread segments contains at most one element, then updating the set of locks associated with the location to the set of locks associated with the thread, and otherwise:

(i) updating the set of locks associated with the location to a set comprising the intersection of (a) the set of locks associated with the location and (b) the set of locks associated with the thread, and

(ii) if the set of locks associated with the location is empty, reporting a warning of a potential race condition.

15. The method of claim 14 wherein the set containing the thread segments in the set of concurrent thread segments that continue to access the location is formed by computing a subset of the set of concurrent thread segments, wherein the subset contains each thread segment *a* that satisfies the following predicate:

for every thread segment *b* in the set of thread segments ordered before *a*, at least one of the following is true: (i) the thread identifier of *a* is not equal to the thread identifier of *b* and (ii) the virtual clock value of *a* is greater than the virtual clock value of *b*.

16. A dynamic race detection system, comprising:

a compiler of a runtime system that inserts calls to a race detector in compiled code; and

a memory allocator of the runtime system that adds to shared memory objects instrumentation information required by the race detector.

17. The system of claim 16 wherein the compiler is a modification of another compiler.

18. The system of claim 16 wherein the memory allocator is an alteration of another memory allocator.

19. A computer-implemented method for dynamic race detection, comprising:

by way of a compiler of a runtime system, inserting calls to a race detector in compiled code; and

by way of a memory allocator of the runtime system, adding instrumentation information required by the race detector to shared memory objects.

20. The method of claim 19 wherein inserting the calls to the race detector is by way of modifying the compiler of the runtime system.

21. The method of claim 20 wherein adding the instrumentation information required by the race detector is by way of changing the memory allocator of the runtime system.